Required Python Libraries: matplotlib, openpyxl

For installation help, see https://www.liquidweb.com/kb/install-pip-windows/. If you already have pip installed, find the directory C:\Users\**username here**AppData\Local\Programs\Python\Scripts. Then hold Shift+Right Click and select open command line or open PowerShell. Then type in pip followed by the name of the Python library (eg, pip install matplotlib). If you are working in PyCharm, locate the Scripts folder associated with your given project and do the same installation process.

There are three versions of this program in the three folders where this document is contained. They all function in the same way so this document applies to all of them.

I. Overview

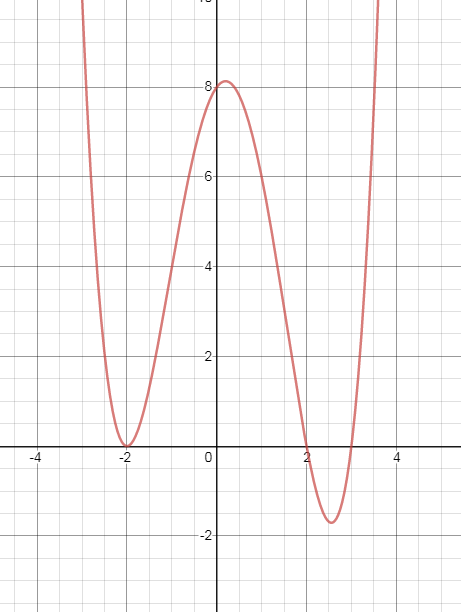
The point cloud analysis and linear regression consists of two separate programs: **data processing.py** (which handles the sorting and sampling of point cloud data) and **lin reg.py** which completes linear regression by comparing point cloud and physical data. Using these programs should require an individual to change variables and copy data.

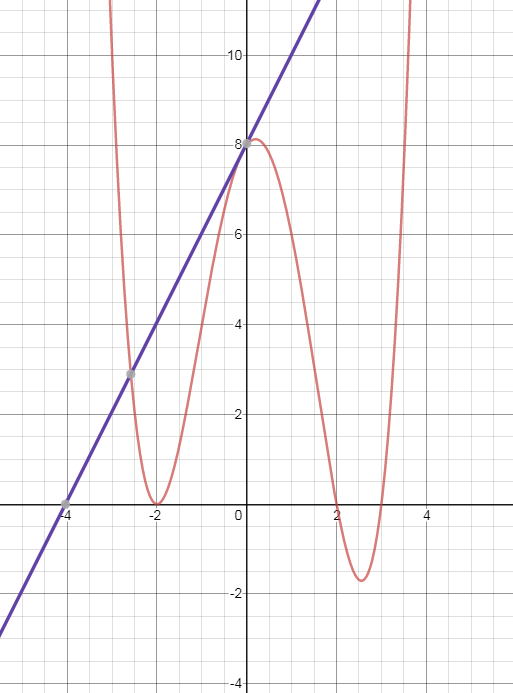
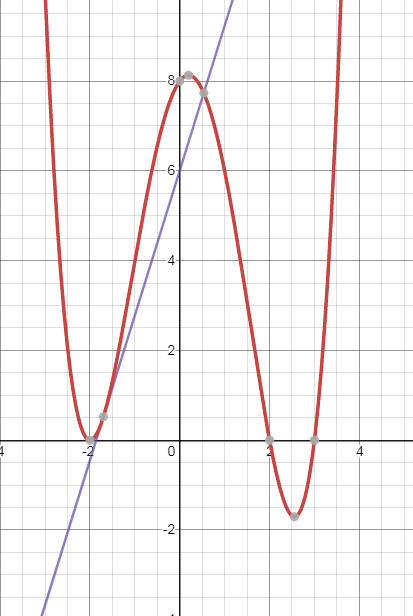
II. Theory

1. Linear Regression

This program finds the local minimum of a very complicated function (in this case, there are 4 input variables). However, the concept of linear regression also applies to functions of lower dimensionality. The main idea is to use the negative gradient of the function at a given point to influence our next guess of the local minimum.

Consider the following function. By calculating the negative gradient (derivative) at an individual point, this shows us which direction to move our guess. For higher dimensional functions, we simply calculate the directional gradient with respect to the variables we can optimize. Then we can re-adjust for multiple variables rather than 1.



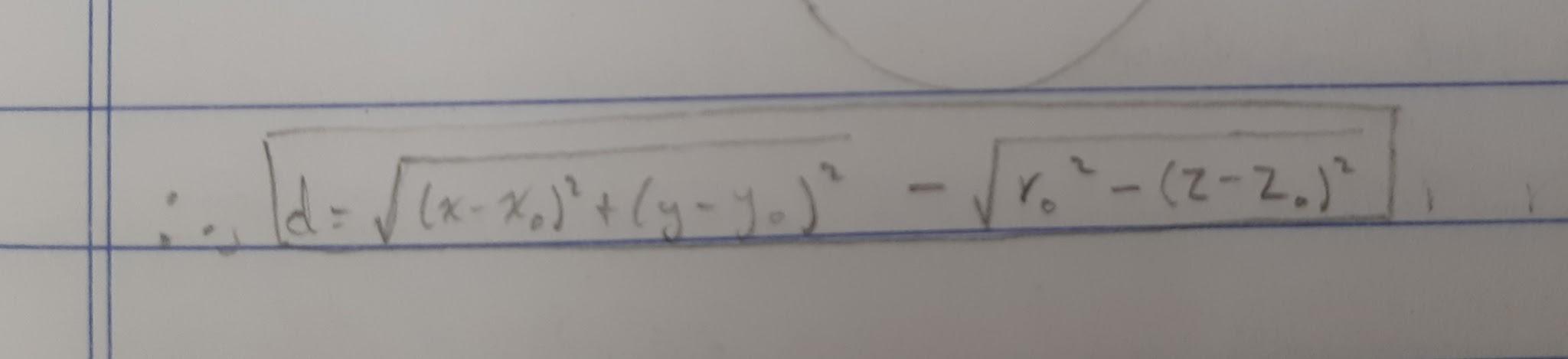


*The gradient (indicated by a tangent line) points away from the direction of a local minima.*

For a more detailed explanation, see this video: https://youtu.be/IHZwWFHWa-w.

1. Geometry

To generate something to optimize, a function was produced that stated the distance from the surface of a sphere non-radially compared with that of a known value; that is to say this function said how bad a given point was. This was a three-parameter function that had the x, y, and z position of a given point as variables and the x, y, z, position and radius of a sphere as constants.



*The final distance equation. The subscript o indicated a constant.*

To generate a loss function, the output of this function was subtracted from the expected distance, squared, summed across all points and divided by the number of points (averaged), and square rooted (normalized so the output is positive).

Then, the directional gradient was analytically solved for the x, y, and z parameters of the center of the sphere as well as the radius (making them variables). The linear regression program uses these analytical solutions and substitutes in guesses for x, y, z, and r to calculate their respective directional gradients. These new values are used to adjust the guesses and the cycle repeats.

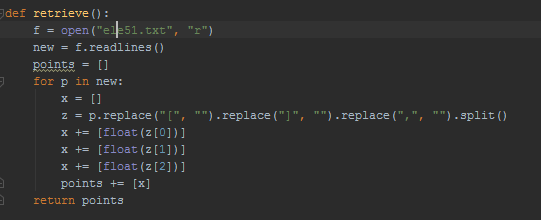
III. Operation

1. Sample the point cloud

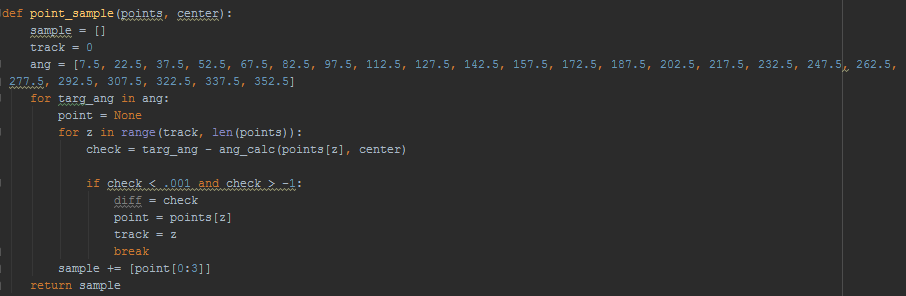
First, sample the point cloud at the desired elevations (in general, 1mm thickness seems to be adequate). Then save the cropped slices to text files and paste them into the same directory as the two Python programs.

1. Sample the slices

Once the point cloud has been sliced at certain elevations, we must sample them at each wicket gate or at any angle necessary. Select the desired file by adjusting the file name associated with the **f** variable in the **retrieve()** function. To change the angles that are sampled, see variable **ang** from the **point\_sample()** function. This list contains the angles from which points are extracted. If you modify this variable, make sure that the angles only go in increasing order (< 360) as points are only checked in clockwise order. This must be repeated for every text file you sampled.



*The retrieve function*

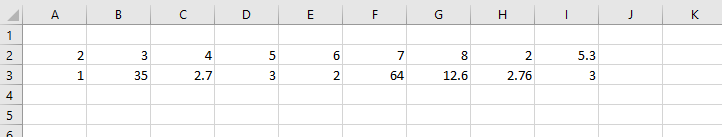


*The point sample function*

For each of the levels that you sample, copy the resulting list to the **lin reg.py** file. Variables with the format **points##** refer to samples from the point cloud (the # symbols refer to a number which corresponds to the elevation). You can adjust the number associated with each variable if you are sampling a different height than what is written in the file, but make sure to change every instance of that variable as well by using Ctrl+F (I don’t bother changing variable names, it doesn’t affect how the program runs. However, make sure to change the file names of the images produced at the end of the file, otherwise there may be some mislabeling problems.).

1. Transfer Excel Data

Take the spreadsheet containing the physical readings corresponding to the point cloud and paste it into the directory where the program **spreadsheet reader.py** is located. Then modify the variables **start** and **end** to indicate from which column to which column data should be taken, **rows** to indicate which rows contain the elevations, and **wb** to change the file name. This should give you lists containing the numbers located in the cells indicated. Copy and paste each of these lists into the corresponding **el##** variables in the **lin reg.py** file (**##** is replaced by an actual number in the program indicating the elevation).



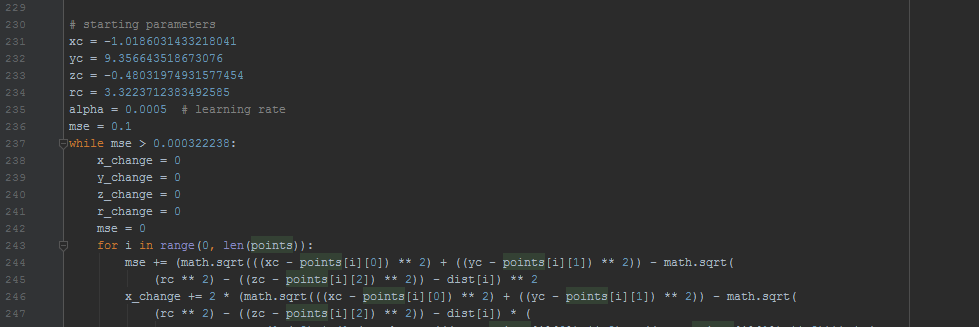
*In this example,* ***start*** *would be A,* ***end***  *would be I, and* ***rows*** *would be [2,3]*

1. Modify variables

If some of the elevations are unused (this means one **points##** and one **el##** variable), see the variables **points** and **dist**. Remove any unused variables from the group of added variables on the right, otherwise you may get errors. These variables are also order sensitive, so make sure the elevations that are used are in the same order for both variables (i.e., the numbers indicating the elevations show up in the same order).

1. Adjust Parameters

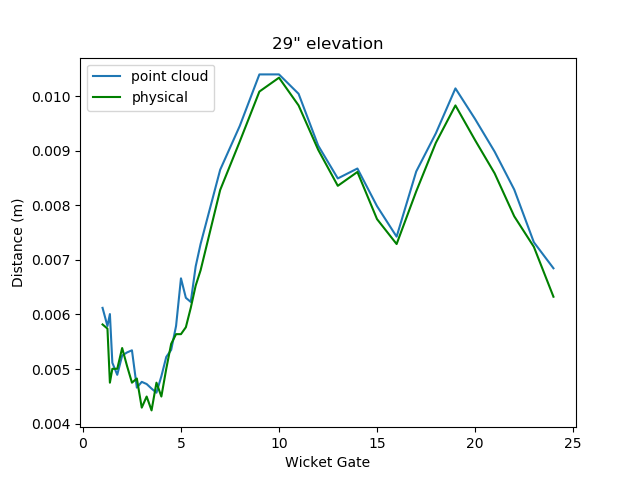
Before running the program, you must adjust 6 starting parameters. These include the four numbers associated with the x, y, z position and radius (from the center) (these are the variables **xc**, **yc**, **zc**, and **rc**), a variable called **alpha**, and the **mse** (the target accuracy) (start at 0.00001 to prevent the program from exiting too early). The xyz variables are in metres, **mse** is in metres, and **alpha** unitless.



*The starting parameters*

**alpha** refers to the learning rate of the program and will likely need to be adjusted over time. It is multiplied by the calculated negative gradient to influence by how much the input parameters are modified. **mse** refers to the accuracy at which the program will stop. It should be selected after seeing what error the program converges to.

Make sure that the x, y, z, and r variables match the data set (a good guess for radius is around 3.3). For **alpha** however, start with a relatively large value like 0.01. After running the program once, you will see that the root mean square error (the number that is on its own line) will not descend any further. At this point, manually stop the program and copy xc, yc, zc, and rc into the starting parameters. Lower **alpha** and run the program again. After maybe one or two cycles, this should give a relatively low **mse** convergence point. Manually stop the program again and copy this **mse** convergence value into the starting **mse** starting parameter. The next time the program runs, it should automatically stop and create a graph for each elevation showing a comparison between the point cloud and excel data.



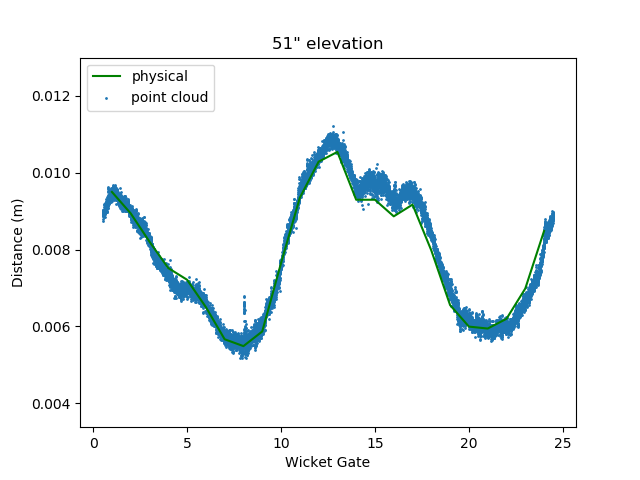
*A sample graph from Unit 1 as found*

1. Final Adjustments and Outliers

If all went according to plan, you should have a few graphs indicating how well the point cloud and physical data match as well as a value for the calculated center, radius, and root mean squared error. In tests I conducted, **mse** finished around 0.4-0.6 mm (0.0004-0.0006 on screen as **mse** is in metres). If it is significantly higher however, I would recommend removing the data from elevations that seem like outliers (i.e., the graphs don’t match very well). This means removing them from the **points** and **dist** variables.

IV. Alternate Program

Another version of this program is available which plots the entire set of sampled points from the corresponding elevation. It functions exactly the same as the program previously described except that the output graphs contain all points that were sampled. It is labelled **lin reg.py** in the unit 3 as found folder.



*A sample graph from Unit 3 as found*